Lecture 4

Game Components
So You Want to Make a Game?

• Will assume you have a design document
  • Focus of next week and a half…
  • Building off the ideas of previous lecture

• But now you want to start building it
  • Need to assign tasks to the team members
  • Helps to break game into components
  • Each component being a logical unit of work.
Traditional Way to Break Up a Game

- **Game Engine**
  - Software, created primarily by programmers

- **Rules and Mechanics**
  - Created by the designers, with programmer input

- **User Interface**
  - Coordinated with programmer/artist/HCI specialist

- **Content and Challenges**
  - Created primarily by designers
Features of Game Engines

- Power the **graphics** and **sound**
  - 3D rendering or 2D sprites

- Power the character and strategic **AI**
  - Typically custom designed for the game

- Power the **physics** interactions
  - Must support collisions at a bare minimum

- Describe the **systems**
  - Space of possibilities in game world
Commercial Game Engines

- Libraries that take care of technical tasks
  - But probably need some specialized code
  - Game studios buy *source code licenses*

- Is XNA a game engine?
  - No AI or physics support at all
  - But external libraries exist (e.g. Box2D)

- Bare bones engine: *graphics + physics*
Game Engines: Graphics

- Minimum requirements:
  - Low level instructions for drawing
  - API to import artistic assets
  - Routines for manipulating images

- Two standard 3D graphics APIs
  - **OpenGL**: Unix, Linux, Macintosh
  - **Direct3D**: Windows

- For this class, our graphics engine is XNA
  - Supports Direct 3D, but will only use 2D
Game Engines: Physics

• Defines physical attributes of the world
  • There is a gravitational force
  • Objects may have friction
  • Ways in which light can reflect

• Does not define precise values or effects
  • The direction or value of gravity
  • Friction constants for each object
  • Specific lighting for each material
Game Engines: Systems

- Physics is an example of a game **system**
  - Specifies the *space of possibilities* for a game
  - But not the *specific parameters* of elements
- Extra code that you add to the engine
  - Write functions for the possibilities
  - But do not code values or when called
- Separates programmer from **gameplay designer**
  - Programmer creates the system
  - Gameplay designer fills in parameters
Systems: Super Mario Bros.

- **Levels**
  - Fixed height scrolling maps
  - Populated by blocks and enemies

- **Enemies**
  - Affected by stomping or bumping
  - Different movement/AI schemes
  - Spawn projectiles or other enemies

- **Blocks**
  - Can be stepped on safely
  - Can be bumped from below

- Mario (and Luigi) can be small, big, or fiery
Traditional RPG Analogy: Engines

- Highest level decisions in the rulebooks
  - Dice mechanisms for entire system
  - Explanation of action types
  - Overview of spell, combat system
  - Statistical requirements for game entities

- SRD: System Reference Document
  - Feature of 3.x D&D (discontinued)
  - Allows creation of compatible games
Modern digital games borrow a lot from traditional RPGs.

- Highest level decisions in the rulebooks
- Dice mechanisms for entire system
- Explanation of action types
- Overview of spell, combat system
- Statistical requirements for game entities
- SRD: System Reference Document
- Feature of 3.x D&D (discontinued)
- Allows creation of compatible games

Parts of a Game
Characteristics of an Engine

• Broad, adaptable, and extensible
  • **Encodes** all *non-mutable* design decisions
  • **Parameters** for all *mutable* design decisions

• Outlines gameplay possibilities
  • Cannot be built independent of design
  • But only needs highest level information

• **Gameplay specification** is sufficient
Data-Driven Design

- No code outside engine; all else is data
  - Purpose of separating system from parameters
  - Create game content with level editors

- Examples:
  - Art, music in industry-standard file formats
  - Object data in XML or other data file formats
  - Character behavior specified through scripts

- Major focus for alpha release
Rules & Mechanics

- Fills in the values for the system
  - Parameters (e.g. gravity, damage amounts, etc.)
  - Types of player abilities/verbs
  - Types of world interactions
  - Types of obstacles/challenges

- But does not include specific challenges
  - Just the list all challenges that could exist
  - Contents of the pallet for level editor
Rules: Super Mario Bros.

- **Enemies**
  - Goombas die when stomped
  - Turtles become shells when stomped/bumped
  - Spinys damage Mario when stomped
  - Piranha Plants aim fireballs at Mario

- **Environment**
  - Question block yields coins, a power-up, or star
  - Mushroom makes Mario small
  - Fire flower makes Mario big and fiery
Traditional RPG Analogy: Mechanics

- Engine + mechanics = core rulebooks
  - Material tailored to genre, setting
  - Less information than an adventure module
  - But enough to create your own adventures

- Vary the mechanics by genre
  - D&D: high fantasy
  - Star Wars: space opera
  - Top Secret: modern spy thriller
Game AI: Where Does it Go?

• Game AI is traditionally placed in mechanics
  • Characters need rules to make right choices
  • Tailor AI to give characters personalities

• But it is implemented by programmer
  • Complicated search algorithms
  • Algorithms should be in game engine

• Holy Grail: “AI Photoshop” for designers
  • Hides all of the hard algorithms
Interfaces

• Interface specifies
  • How player does things  (player-to-computer)
  • How player gets feedback (computer-to-player)

• More than engine+mechanics
  • They just describe what the player can do
  • Do not specify how it is done

• Bad interfaces can kill a game
Interface: Dead Space
Traditional RPG Analogy: Interface

- Interface includes:
  - Character sheets
  - Pencils
  - Maps
  - Dice
  - Player voices

- Alternate interfaces for D&D
  - LARPing
  - Play-by-mail
Interface Tips

- Must consider input devices in design
  - For PC, typically mouse and keyboard
  - Game controllers have different “feel”

- Consider depth and width of interface
  - Details are best processed at the center of vision
  - Peripheral vision mostly detects motion

- Strive for “invisible” interface (metaphorically)
  - Familiarity is better than innovation
Content and Challenges

• Content is **everything else**

• **Gameplay** content define the actual game
  • Goals and victory conditions
  • Missions and quests
  • Interactive story choices

• **Non-gameplay** content affects player experience
  • Graphics and cut scenes
  • Sound effects and background music
  • Non-interactive story
Traditional RPG Analogy: Content

- Content is what creates an adventure
  - Could include adventure modules
  - But also includes the DM’s imagination
    - “Dealing with the exceptions” 90% of time
    - DM must quickly adapt to the players

- Ability to improvise provides another lesson:
  - Content should be easy to change as needed
  - Needs well-designed engine+mechanics+interface
Why the division?

- They are not developed sequentially
  - Content may requires changes to game engine
  - Interface is changing until the very end
- Intended to organize your design
  - **Engine**: decisions to be made early, hard-code
  - **Mechanics**: mutable design decisions
  - **Interface**: how to shape the user experience
  - **Content**: specific gameplay and level-design
Milestones Suggestions

<table>
<thead>
<tr>
<th>Nondigital</th>
<th>Gameplay</th>
<th>Technical</th>
<th>Alpha</th>
<th>Beta</th>
<th>Release</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-Engine Tech</td>
<td>Completed Game Engine</td>
<td>Mechanics (Implementation)</td>
<td>Interface (Functional Mock-up)</td>
<td>Interface (Polishing)</td>
<td>Content</td>
</tr>
</tbody>
</table>

Design Elements
Summary

- Game is divided into four components
  - Should keep each in mind during design
  - Key for distributing work in your group
- But they are all interconnected
  - System/engine limits your possible mechanics
  - Content is limited by the type of mechanics
- Once again: **design is iterative**